FLORIDA SOUTHERN COLLEGE

Data Structures
CSC 3280

Fall 2021

Program 3: *FSC Recurse*
Assigned: Thursday, October 5, 2021
Due: Thursday, October 14, 2020 by 11:59 PM

Purpose:

1. Make use of recursion to solve problems.

Read Carefully:

- This program is worth 7% of your final grade.

- **WARNING**: This is an individual assignment; you must solve it by yourself. Please review the definition of cheating in the syllabus, the FSC Honor Code, and the serious consequences of those found cheating.
    - **The FSC Honor Code pledge MUST be written as a comment at the top of your program**.

- When is the assignment due? The date is written very clearly above.
    - **Note:** once the clock becomes 11:59 PM, the submission will be closed! Therefore, in reality, you must submit by 11:58 and 59 seconds.

- LATE SUBMISSION: you are allowed to make a late submission according to the rules defined in the syllabus. Please see course syllabus for more information.

- **Canvas Submission**:
    - This assignment must be submitted online via Canvas.
    - You should NOT make a package for this assignment.
    - Further, there is only one file. Therefore, you should NOT zip that file for submission.
    - On Canvas, you should merely submit your single Java file:
        - `FSCrecurse.java`

- **Coding Rooms Submission**:
    - This assignment must also be submitted at Coding Rooms in order to receive credit.

Program 3: FSC Recurse

The Problems

Below are a series of problems you need to solve using **recursive** methods. You will write a program that will read commands from user input, with each command referring to one of the recursive problems to be executed. Each command will be followed (on the same line of input) by the respective parameters required for that specific problem.

Four Problems to Solve Using Recursion:

(1) MocsMath

Write a recursive method that sums up the values of all factorials from 0 to n and returns the result (as a double). For example, if n is 5, the method should return 0! + 1! + 2! + 3! + 4! + 5! So the method should return 154. Inside your method, you should call the recursive factorial method (from slides) to calculate the individual factorials. This one problem will have three methods associated with it, the wrapper method (see explanation on wrappers below), the recursive method, and even the recursive factorial method.

(2) MocsShape

Simply write a recursive method that prints a partial triangle made of small stars ("* "). The method will have two parameters, ***m*** and ***n***. It is guaranteed that 1 <= ***m*** <= ***n***. The following is an example where m=3 and n = 5. The first line printed has 3 stars, then 4 starts, and then 5 stars. Then, the pattern is repeated backwards, going from ***n*** to ***m***.

```
* * *
* * * *
* * * * *
* * * * *
* * * *
* * *
```

***Hint: you CAN use a for loop to draw one, single row of the shape. But you need recursion to call the method on the "smaller" version of the problem.

(3) MocsGame

The game starts with ***n*** dollars. The goal of the game is to follow rules (shown below) to see if you can end up with exactly 42 dollars. If you can do this, then you get to keep all the money!

Here are the rules:

- If *n* is even, then you may choose to give back exactly *n*/2 dollars (integer division).
- If *n* is divisible by 3 or 4, then you may choose to multiply the last two digits of *n* and then give back this many dollars.
  (Hint: *n* %10 gives you the last digit and (*n* %100)/10 gives you the 2nd to last digit.
- If *n* is divisible by 5, then you may choose to give back exactly 42 dollars.

Again, the goal is to give back dollars and finish with exactly 42 dollars.

Example: you start with 250 dollars. You can do the following steps to arrive at 42 dollars:

- Because 250 is divisible by 5, you could give back 42 dollars, leaving you with 208 dollars.
- Because 208 is even, you could return half of the remaining dollars, leaving you with 104 dollars.
- Because 104 dollars is even, you could again return half of the remaining dollars, leaving you with 52 dollars.
- Because 52 is divisible by 4, you could multiply the last two digits (2*5 = 10) and return that amount (10). This leaves you with exactly 42 dollars!

Note: often you can make many choices. For example, if *n* is 180, this means that *n* is even, *n* is divisible by both 3 and 4, and *n* is also divisible by 5. This means that you can use any of the rules to reduce your money. You can return exactly *n*/2 dollars, you can return *k* dollars, where *k* is the product of the last two digits, or you can even give back exactly 42 dollars! So the question is, which should you do? (Hint: this is computer science, you try everything)

Your job is to write a method, **MocsGame**, that will return true if you can win the game (if you can arrive to exactly 42 dollars) or false otherwise.

(4) MocsHop

FSC students have so much free time on their hands that they drew a big jumping game outside the entrance to the Cube. There are very large numbers written on the ground as follows:

$\boxed{4}$ 4 1 5 2 6 3 4 2 0

The number, with a square around it, indicates where you are currently standing. You can jump left or right down the line by jumping the number of spaces indicated by the number you are standing on. So if you are standing on a 4, you can jump either left 4 spaces or right 4 spaces. <u>*BUT: You cannot jump past either end of the line</u>.

For example, the first number (4) only allows you to jump right, since there are no numbers to the left that you can jump to.

<u>**The goal**</u>: <u>you want to get to the 0 at the far end (right side) of the line</u>. You are also guaranteed that there will be only one zero, which, again, will be at the far right side.

Here is how you do that with the above line:

Starting position	$\boxed{4}$ 4 1 5 2 6 3 4 2 0
Step 1: Jump right	4 4 1 5 $\boxed{2}$ 6 3 4 2 0
Step 2: Jump left	4 4 $\boxed{1}$ 5 2 6 3 4 2 0
Step 3: Jump right	4 4 1 $\boxed{5}$ 2 6 3 4 2 0
Step 4: Jump right	4 4 1 5 2 6 3 4 $\boxed{2}$ 0
Step 5: Jump left	4 4 1 5 2 6 $\boxed{3}$ 4 2 0
Step 6: Jump right	4 4 1 5 2 6 3 4 2 $\boxed{0}$

Some MocsGame lines have multiple, correct paths to 0 from the given starting point. Other lines have no paths to 0, such as the following:

```
3 1 2 3 0
```

In this line, you can jump between the 3's, but not anywhere else. So this will return false.

You are to write a recursive method that will return a boolean (true or false) to show if we can solve the game (if we can get to the rightmost 0). If we can get to the zero, then your method should return true. If we cannot get the rightmost 0, then the method should return false.

NOTE: you MUST study the detailed input and output for this problem (and every problem) to make sure you understand what the recursion is doing.

Implementation

Each recursive method MUST have a wrapper method enclosing it where you will do input/output file processing as well as calling the actual recursive method.

From Wikipedia:

"A wrapper function is a function in a computer program whose main purpose is to call a second function".

As an example, here is one of the wrapper methods you will use:

```
void MocsMath(Scanner input)
```

This would be the wrapper method for one of the recursive problems, called **`MocsMath()`** (described, in detail, above). These wrapper methods will simply do three things:

1. deal with the processing of the input file
2. most importantly, the wrapper method will make the initial call to your recursive method that will actually solve the problem.
3. Finally, the wrapper methods will print to the output. Note: printing to the output may also occur in the actual recursive methods.

Of course, at your own discretion (meaning, your own choice), you can make additional methods for your recursive methods to use, such as methods to swap values in an array, take the sum of an array of values, printing methods, etc.

> ***__HINT__**: We have posted an example program with a small recursive method that uses a wrapper method as described above.

<u>Wrapper Methods</u>

As mentioned, you MUST use the following three wrapper methods <u>EXACTLY</u> as shown:

```
    public static void MocsMath(Scanner in)
    public static void MocsShape(Scanner in)
    public static void MocsGame(Scanner in)
    public static void MocsHop(Scanner in)
```

**This means you will make the Scanner variable, inside main (to read the command from the user), and then you will send that Scanner variable to the four wrapper methods.*

<u>Input & Output Specifications</u>

- <u>File IO will *not* be used for this program</u>. Allow me to repeat: you will *not* read from a file, nor will you write to a file for this program. You will read from the user via System.in and you will print to the standard output (the console).
- That said, sample input and output files have been provided to show you possible input (from the user) and the matching, expected output.
- The first line of the input will have one positive integer, representing the number of commands (lines) that the user will be inputting.
- Each of the following n lines will have a command, and each command will be followed by appropriate data as described below (and this data will be on the same line as the command).

The commands (for the four recursive methods), and their relevant data, are described below:

`MocsMath` – This command will be followed on the same line by one positive integer, *n*, as described above.

`MocsShape` – This command will be followed on the same line by two positive integers, *m* and *n*, as described above.

`MocsGame` – This command will be followed by one positive integer, *n*, as described above.

`MocsHop` – This command will be followed by an integer, **`start`**, representing the initial position on the line you are playing on (0 means the far left, the first position); **`size`**, the number of integers drawn on the ground; and then `size` will be followed by `size` number of integers, the integers drawn on the ground. (see sample input for examples)

*****WARNING*****

Your program MUST adhere to the EXACT format shown in the sample output (spacing capitalization, use of dollar signs, periods, punctuation, etc). As always, the grading files will be large and comprehensive, resulting in very large output files. As such, we will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade. Again, your output MUST ADHERE EXACTLY to the sample output.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.
2) Your algorithmic design.
3) Correctness.
4) **Use of Recursion. If your program does not use recursion, you will get a zero.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
   ***Please RE-READ the above note on comments.
6) Compatibility to the **newest version** of IntelliJ. (If your program does not compile in IntelliJ, you will get a large deduction from your grade.)
7) Your program should include a header comment with the following information: your name, **email**, date, AND HONOR CODE.
8) Your output MUST adhere to the EXACT output format shown in the sample output.

Deliverables

You should submit a single java file: **FSCrecurse.java**

***READ the following carefully

Do NOT use a package for this program. Do NOT submit a zip file. There is only one Java file for this program. Therefore, a zip is therefore unnecessary. Points will be deducted if you submit a zip or if you use a package.

Suggestion: START EARLY!